R-Skript PUNO-Forschungsprojekt

Teil 4 – Grafiken mit `ggplot2`

Dominik Vogel

Stand: 03.04.2019

Inhaltsverzeichnis

1 Grundlagen und Begriffe

`ggplot2` baut Grafiken grundsätzlich aus verschiedenen Ebenen („Layers“) auf. Ein Layer kann grundsätzlich jedes grafische Element, wie das Koordinatensystem, Balken, Punkte, Linien usw. sein. Dies bietet den Vorteil, das jedes Element einzeln angesprochen und auch modifiziert werden kann.

Ausführliche Erläuterungen zu `ggplot2` finden sich in Kapitel 4 von Field et al. (2012) sowie in Wickham, H. (2016): ggplot2: Elegant Graphics for Data Analysis. New York: Springer.

Laden wir zunächst das `tidyverse` Paket:

```
library(tidyverse)
```

`ggplot2` ist ein Teil des Pakets `tidyverse`. Wir müssen es also nicht einzeln laden.

Die Darstellung wird etwas publikationsfreundlicher, wenn wir zusätzlich das Paket `cowplot` laden:

```
install.packages("cowplot", dep = TRUE)
library(cowplot)
```

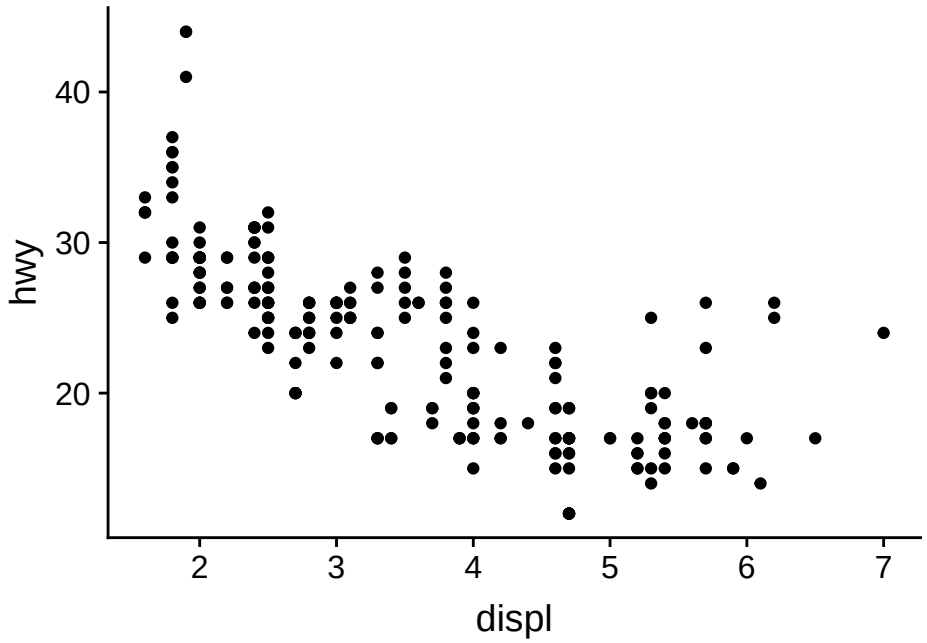
1.1 Grundstruktur

Eine Grafik hat in `ggplot2` drei Kernkomponenten:

1. **Daten**
2. Ein Set von s.g. **aesthetic mappings**, also eine Verknüpfung von Daten und visuellen Eigenschaften
3. Mindestens ein **Layer**, der beschreibt, wie die einzelnen Beobachtungen dargestellt werden sollen.

Im Folgenden können wir das an einem Beispiel betrachten. Wir nutzen dafür den Datensatz `mpg`, ein Beispieldatensatz von `ggplot2`, der Daten zum Benzinverbrauch verschiedener Autos zur Verfügung stellt und automatisch mit `tidyverse` geladen wird:

```
ggplot(mpg,
       aes(y = hwy, x = displ)) +
    geom_point()
```



1. Daten: Der Datensatz (`mpg`) wird unmittelbar nach dem Grundbefehl `ggplot` angegeben.
2. Aesthetic mapping (`aes()`): Benzinverbrauch auf die y-Achse (`y = hwy`) und Hubraum auf die x-Achse (`x = displ`) gemappt.
3. Layer: Punkte (`geom_point`) nach einem `+` an den Grundbefehl angefügt.

1.2 Geoms

Die grundlegende Form einer Grafik wird durch ein *Geom* definiert. Im oberen Beispiel haben wir mit `geom_point()` einen Scatterplot definiert. Es gibt aber noch viele weitere Geoms. Die gebräuchlichsten sind:

- `geom_bar()`: Balkendiagramm
- `geom_point()`: Scatterplot
- `geom_line()`: Verknüpfung von Datenpunkten mit einer Geraden.
- `geom_smooth()`: „Smoother“ (z. B. eine Regressionsgerade)
- `geom_histogram()`: Histogram
- `geom_boxplot()`: Boxplot
- `geom_text()`: Text (i.d.R. zusätzlich zu einem anderen Geom. Zum Beispiel Beschriftungen von Datenpunkten)
- `geom_density()`: Density plot (wie zum Beispiel eine Kerndichtefunktion in einem Histogramm)
- `geom_errorbar()`: Fehlerbalken (z. B. Konfidenzintervalle)
- `geom_hline()` und `geom_vline()`: Nutzerdefinierte horizontale oder vertikale Linie

1.3 Aesthetics

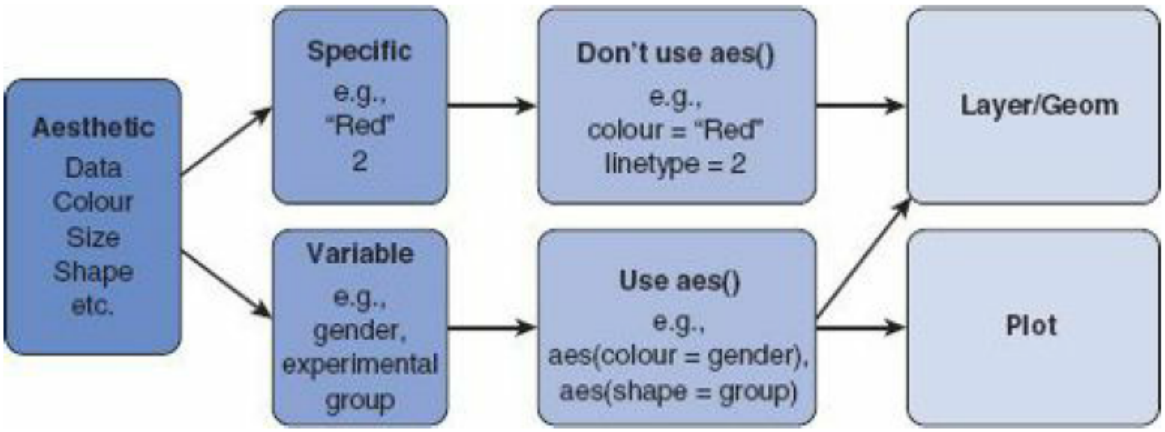
Aesthetics steuern das Erscheinungsbild von Elementen innerhalb eines Geoms. Dies kann in zwei Formen geschehen. Entweder für die gesamte Grafik, dann wird das Aesthetics in der ersten Zeile innerhalb von `ggplot()` angegeben, oder für ein einzelnes Geom, dann wird es innerhalb von `geom_XXX()` angegeben.

Für die verschiedenen Geoms müssen immer einer oder mehrere Aesthetics zwangsweise angegeben werden. Dies ist jeweils die Angabe, welche Daten wie dargestellt werden sollen (z. B. x-Koordinate und y-Koordinate). Dazu stehen weitere Einstellungen wie Farbe, Größe oder Symbol zur Verfügung. Field et al. (2012) geben in Kapitel 4.3.2 einen Überblick über erforderliche und optionale Aesthetics für die wichtigsten Geoms.

1.3.1 Spezifische und variable Aesthetics

Aesthetics unterscheiden sich darin, ob sie auf einen festen Wert definiert werden oder als Funktion einer Variable. Ein fester Wert, ist zum Beispiel die Farbe rot (`color = "red"`). Ein variabler Wert wäre die Farbe in Abhängigkeit des Geschlechts (`color = gender`).

Je nach dem, ob ein Aesthetic spezifisch oder variabel ist, unterscheidet sich der Befehlsaufbau. Spezifische Aesthetics werden ohne `aes()` verwendet (z. B. `color = "red"`). Variable Aesthetics erfordern hingegen die Verwendung von `aes()` (z. B. `aes(color = gender)`). Eine gute Übersicht hierzu liefern Field et al. (2012):



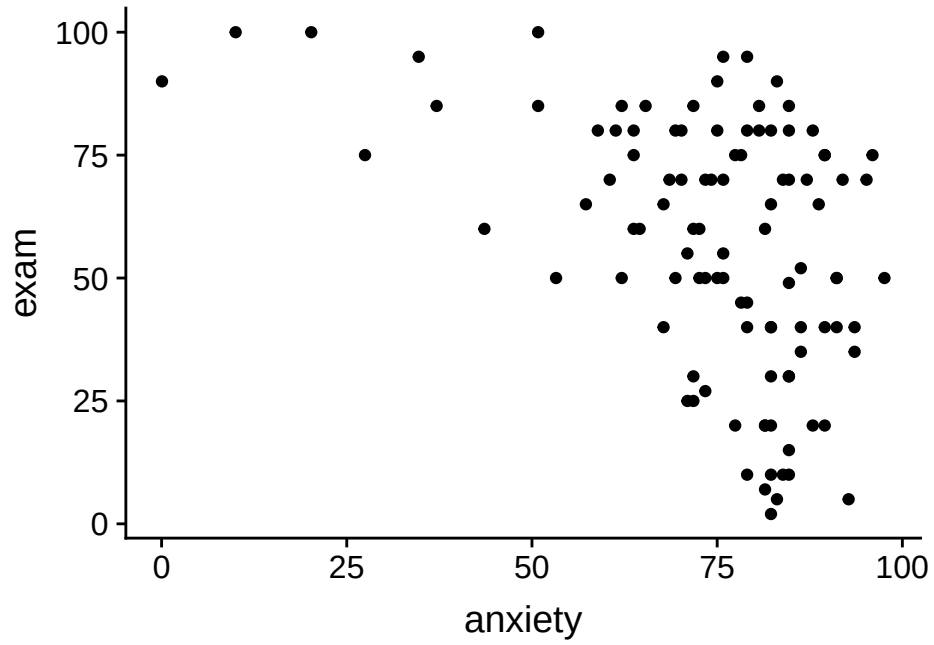
2 Scatterplot

Beginnen wir unseren kurzen Überblick über die wichtigsten Grafiken mit dem Scatterplot. Nutzen wir hierfür die Exam Anxiety Daten, die wir bereits kennen:

```
exam <- read_csv("data/exam_anxiety.csv")
```

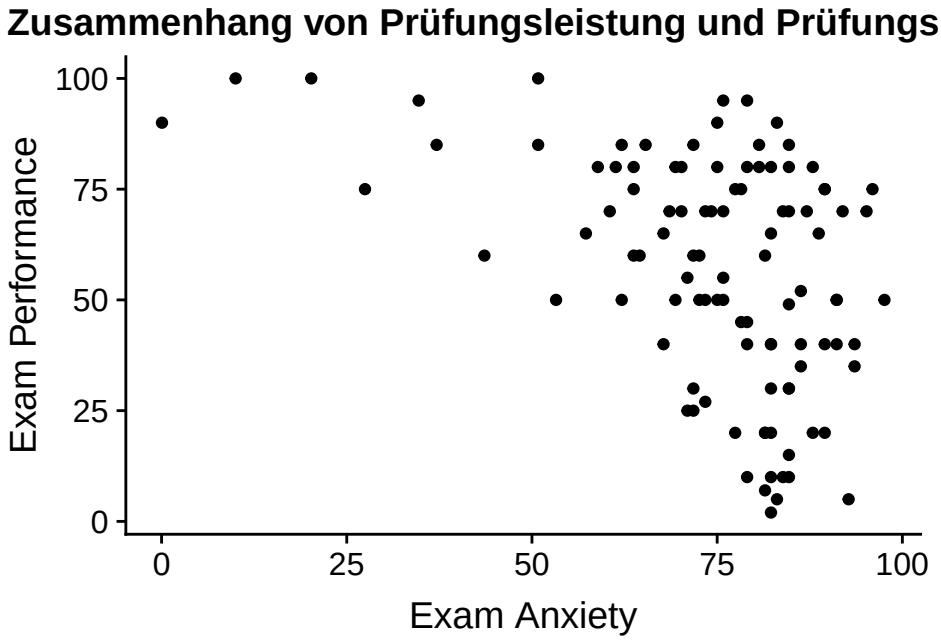
```
## Parsed with column specification:
## cols(
##   code = col_double(),
##   revise = col_double(),
##   exam = col_double(),
##   anxiety = col_double(),
##   gender = col_character()
## )
```

```
scatter <- ggplot(exam, aes(x = anxiety, y = exam)) +
  geom_point()
scatter
```



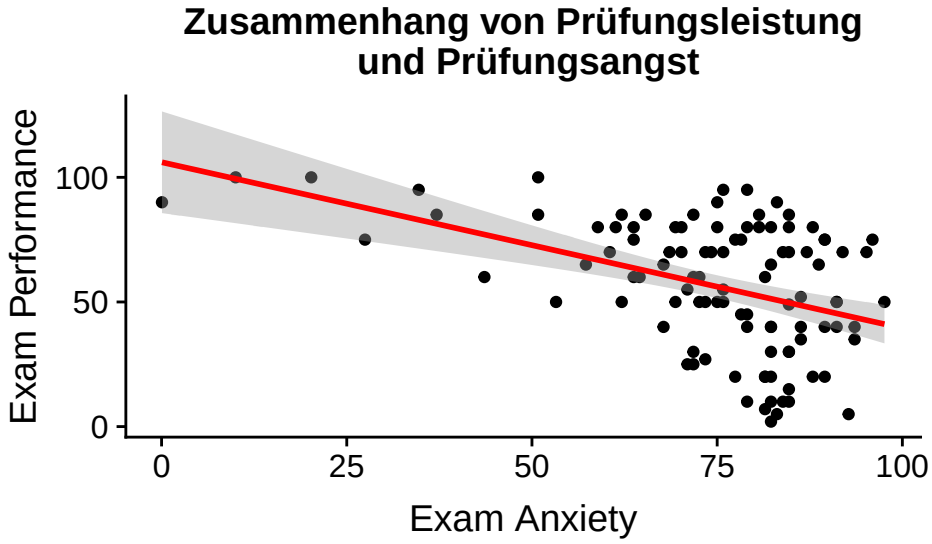
Mit dem Layer `labs` können wir die Beschriftung der Achsen sowie den Titel ändern.

```
scatter <- ggplot(exam, aes(x = anxiety, y = exam)) +
  geom_point() +
  labs(x = "Exam Anxiety",
       y = "Exam Performance",
       title = "Zusammenhang von Prüfungsleistung und Prüfungsangst")
scatter
```



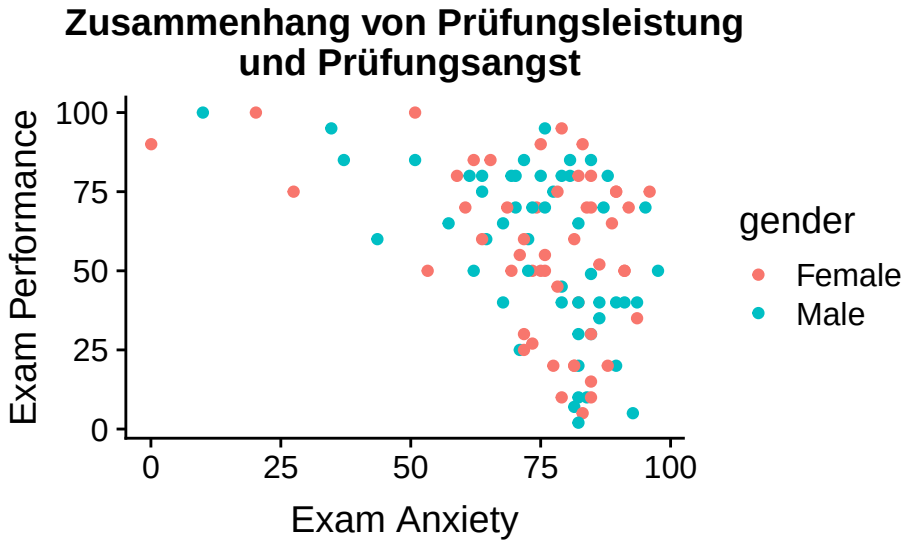
Zu einem Zusammenhang möchten wir natürlich auch noch eine Regressionsgerade hinzufügen. Mit `geom_smooth` ist dies kein Problem (`geom_smooth(method = "lm", color = "red")`:

```
scatter <- ggplot(exam, aes(x = anxiety, y = exam)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red") +
  labs(x = "Exam Anxiety", y = "Exam Performance",
       title = "Zusammenhang von Prüfungsleistung\n und Prüfungsangst")
scatter
```



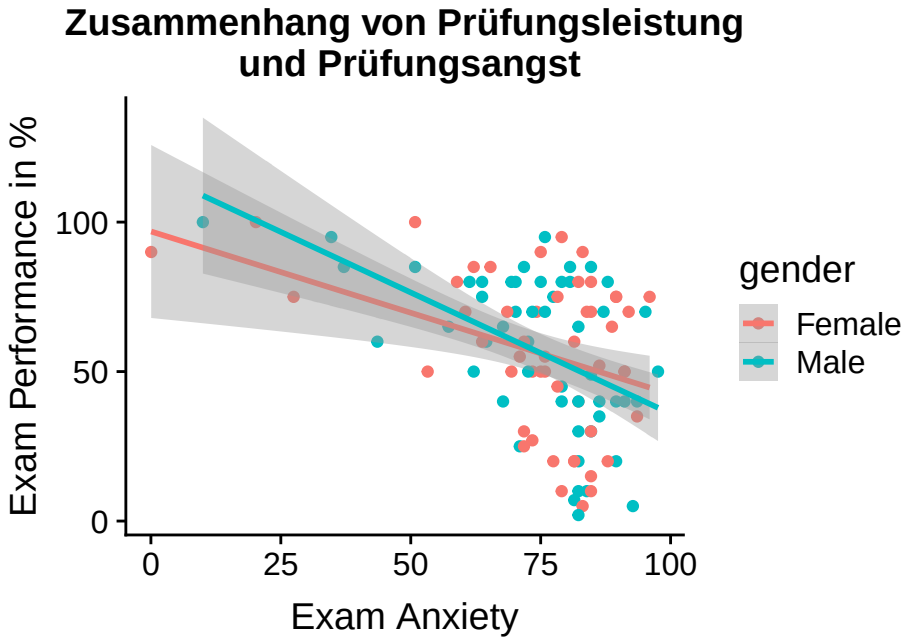
Zusätzlich wäre möglicherweise der Unterschied zwischen Männern und Frauen interessant. Hierfür könnten wir die einzelnen Punkte nach Geschlecht einfärben (`geom_point(aes(color = gender))`):

```
scatter <- ggplot(exam, aes(x = anxiety, y = exam)) +
  geom_point(aes(color = gender)) +
  labs(x = "Exam Anxiety", y = "Exam Performance",
       title = "Zusammenhang von Prüfungsleistung\n und Prüfungsangst")
scatter
```



Bringen wir nun noch Regressionsgerade und Geschlechterunterschiede zusammen und fügen zwei Regressionsgeraden für die Geschlechter hinzu. Hierfür fügen wir zu `geom_smooth` einen Aesthetic hinzu (`geom_smooth(method = "lm", aes(fill = gender))`).

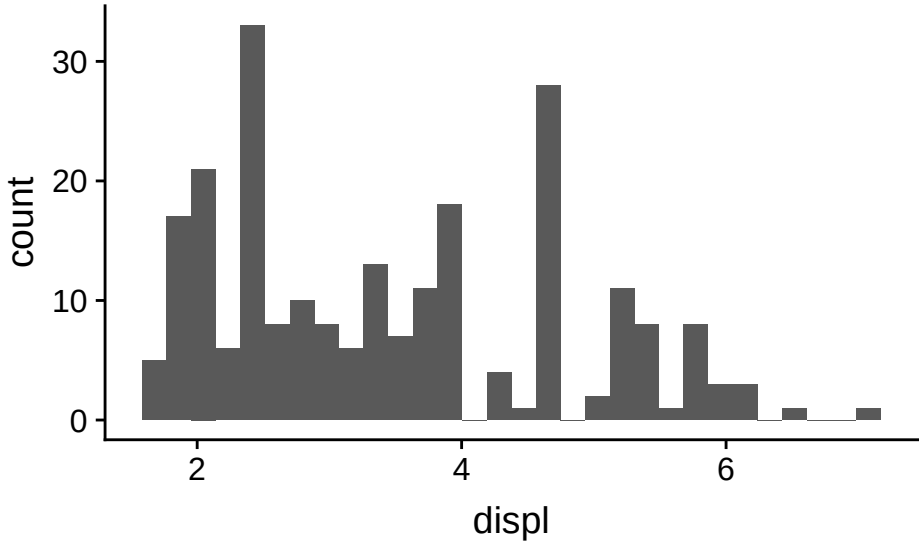
```
scatter <- ggplot(exam, aes(x = anxiety, y = exam)) +
  geom_point(aes(color = gender)) +
  geom_smooth(method = "lm", aes(color = gender)) +
  labs(x = "Exam Anxiety",
       y = "Exam Performance in %",
       title = "Zusammenhang von Prüfungsleistung\n und Prüfungsangst")
scatter
```



3 Histogramm

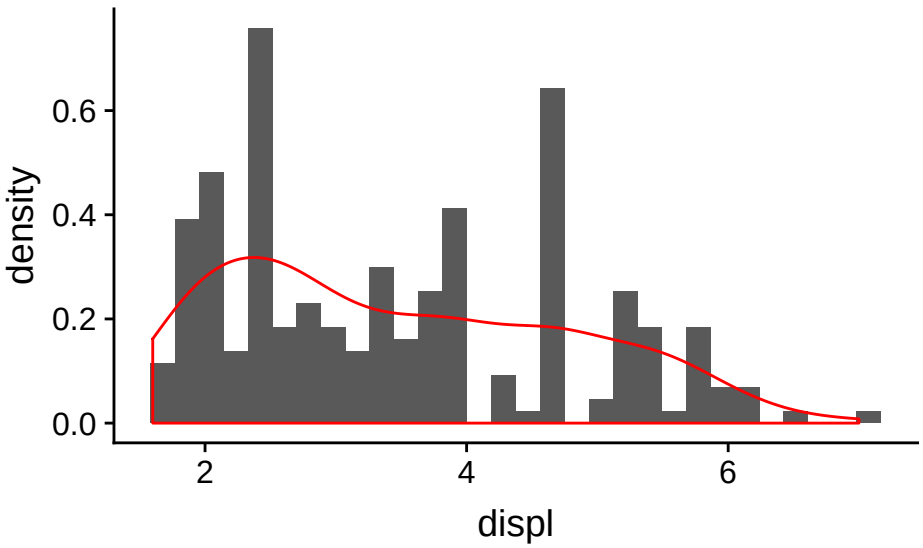
Zur Erstellung eines Histogramms steht das Geom `geom_histogram()` zur Verfügung:

```
hist <- ggplot(mpg, aes(x = displ)) +
  geom_histogram()
hist
```



Zu diesem Histogramm können wir auch eine Kerndichtefunktion hinzufügen. Dazu geben wir für das Aesthetic an, dass die Werte als Dichtefunktion dargestellt werden sollen: `aes(x = displ, stat(density))`. Zusätzlich fügen wir ein neues Geom `geom_density()` hinzu.

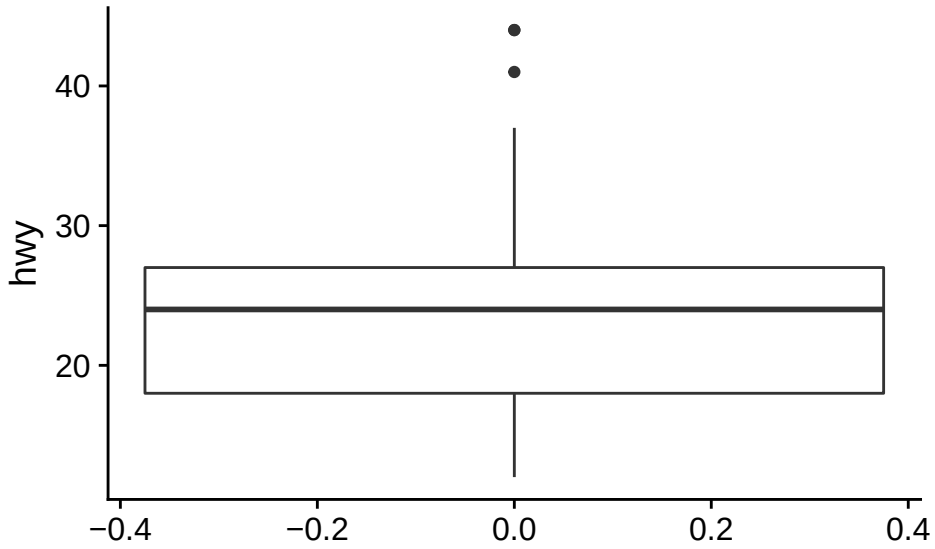
```
hist <- ggplot(mpg, aes(x = displ, stat(density))) +
  geom_histogram() +
  geom_density(color = "red")
hist
```



4 Boxplot

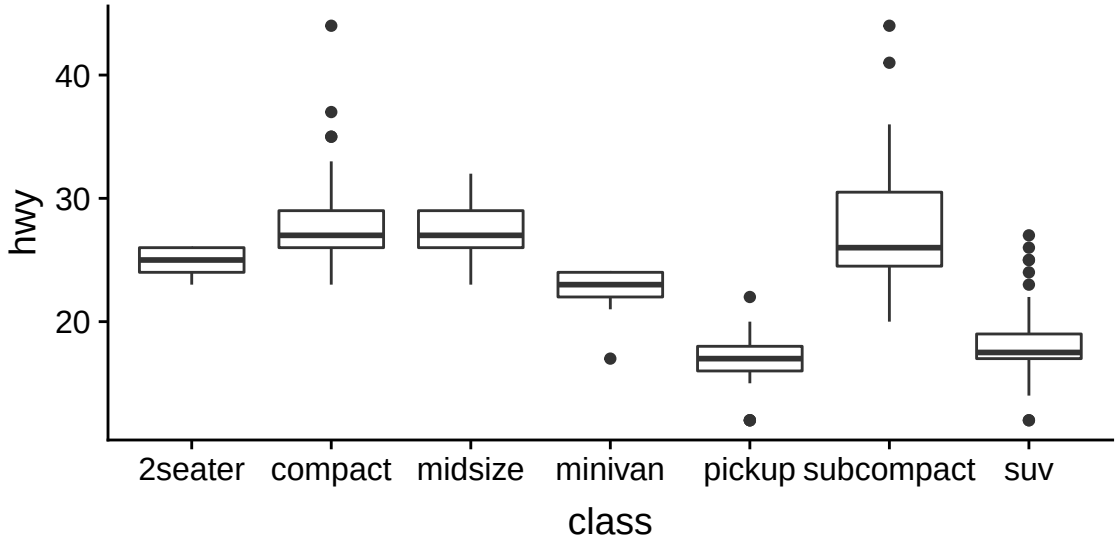
Für Boxplots ist das Geom `geom_boxplot()` zuständig.

```
boxplot <- ggplot(mpg, aes(y = hwy)) +
  geom_boxplot()
boxplot
```



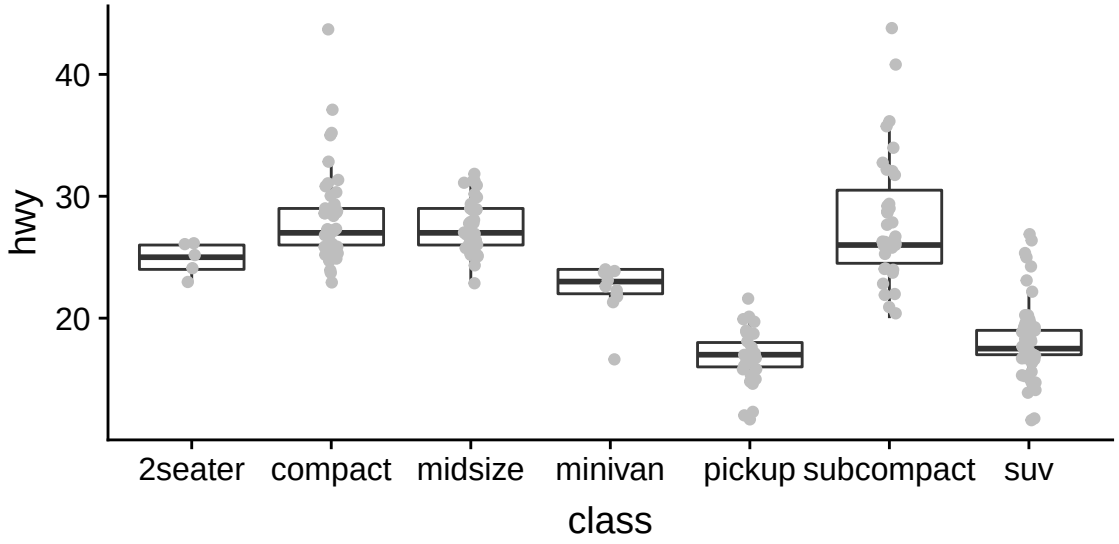
Gruppenunterschiede können ebenfalls sehr einfach dargestellt werden:

```
boxplot <- ggplot(mpg, aes(y = hwy, x = class)) +
  geom_boxplot()
boxplot
```



Um dem/der Betrachter*in einen umfassenden Eindruck von der Verteilung der Daten zu geben, empfiehlt es sich, nicht nur die Ausreißer, sondern alle Datenpunkte einzublenden. Hierfür können wir das Geom `geom_jitter` verwenden. Um die Ausreißer nicht doppelt zu plotten, fügen wir bei `geom_boxplot` die Option `outlier.shape = NA` hinzu.

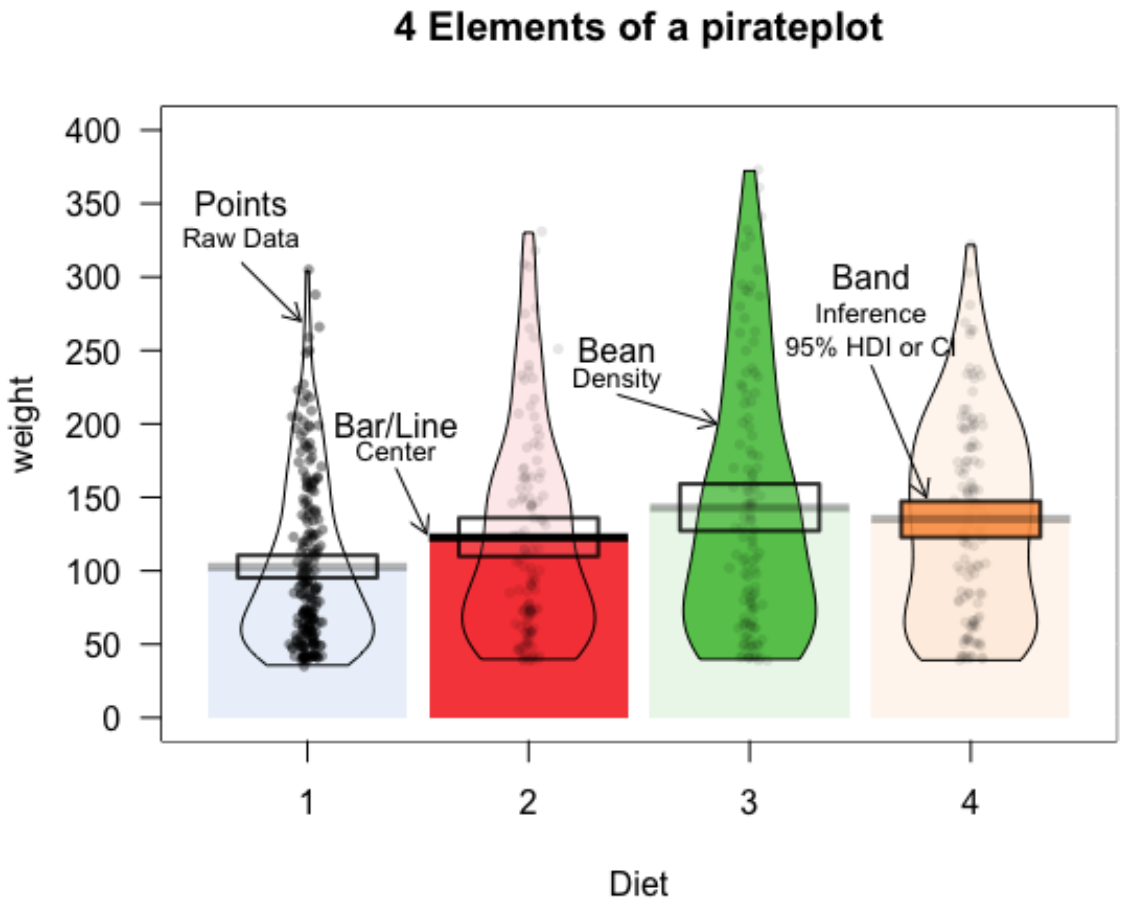
```r
boxplot <- ggplot(mpg, aes(y = hwy, x = class)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(width = 0.05, color = "gray")
boxplot
```



5 Pirateplot

Der „Pirateplot“ aus dem Paket `yarrr` ist eine Visualisierung, die umfassende Informationen über die Daten bereitstellt. Nathaniel Phillips, der Entwickler des Pirateplots beschreibt ihn wie folgt (https://cran.r-project.org/web/packages/yarrr/vignettes/pirateplot.html):

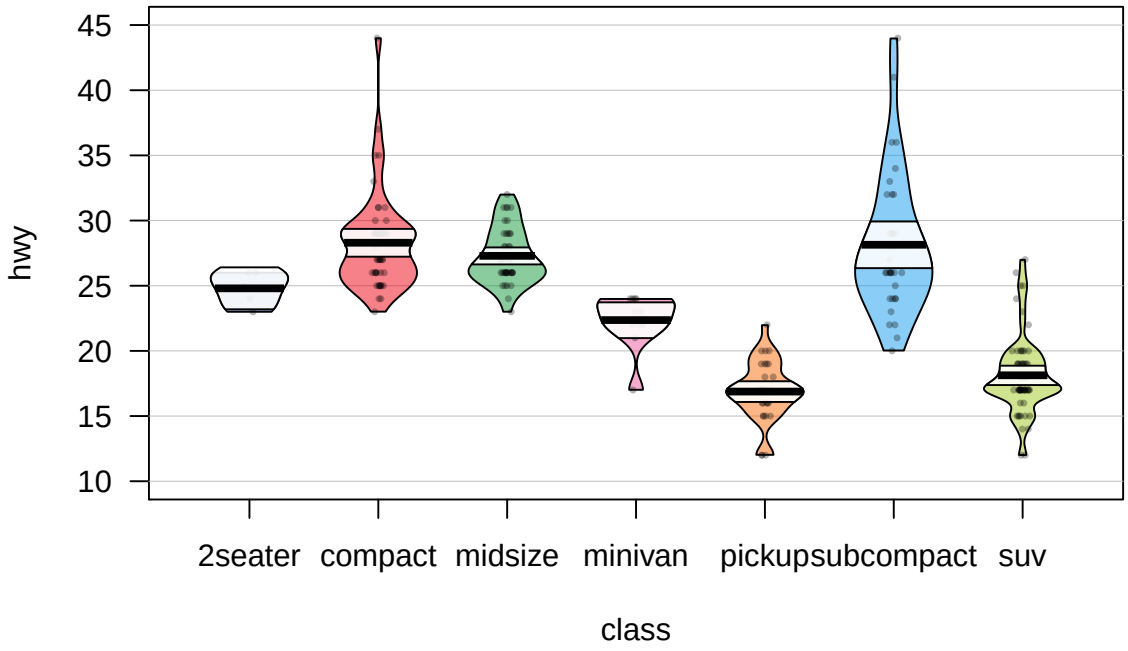
> A pirateplot, is the RDI (Raw data, Descriptive statistics, and Inferential statistics) plotting choice of R pirates who are displaying the relationship between 1 to 3 categorical independent variables, and one continuous dependent variable.
>
> A pirateplot has 4 main elements
>
> 1. points, symbols representing the raw data (jittered horizontally)
> 2. bar, a vertical bar showing central tendencies
> 3. bean, a smoothed density (inspired by Kampstra and others (2008)) representing a smoothed density
> 4. inf, a rectangle representing an inference interval (e.g.; Bayesian Highest Density Interval or frequentist confidence interval)



Ein Pirateplot ist schnell erstellt:

```
library(yarrr)
```

```
pirateplot(hwy ~ class,
           data = mpg,
           inf.method = "ci", # Konfidenzintervalle statt HDI
           theme = 3)
```



6 Balkendiagramm

Ein kleiner Hinweis vorweg: Balkendiagramme sind keine ideale Visualisierung von Daten, da sehr unterschiedliche Daten dieselben Balkendiagramme produzieren können. Es gibt daher inzwischen viele Experten, die fordern, keine Balkendiagramme in wissenschaftlichen Publikationen zu nutzen (https://simplystatistics.org/2019/02/21/dynamite-plots-must-die/). Als Alternativen bieten sich Boxplots oder auch der Pirateplot (s.o.) an. Bei relativ wenigen Beobachtungen, wie wir sie bei Experimenten häufig haben, bietet es sich auch an, zusätzlich die Daten zu plotten.

Für ein Balkendiagramm, nutzen wir das Geom `geom_bar`. Dieses erwartet allerdings bereits aufbereitete Daten, die in einer Variable verschiedenen Gruppen und in der anderen die dazugehörigen Werte speichert. Wenn man, wie im folgenden Beispiel, den Mittelwert verschiedener Variablen in `Lecturer_Data` darstellen möchte, benötigt man also aufbereitete Daten. Um dies nicht händisch erledigen zu müssen, verwenden wir die Funktion `stat_summary()`. Diese Funktion ist folgendermaßen aufgebaut:

```
stat_summary(function = x, geom = y)
```

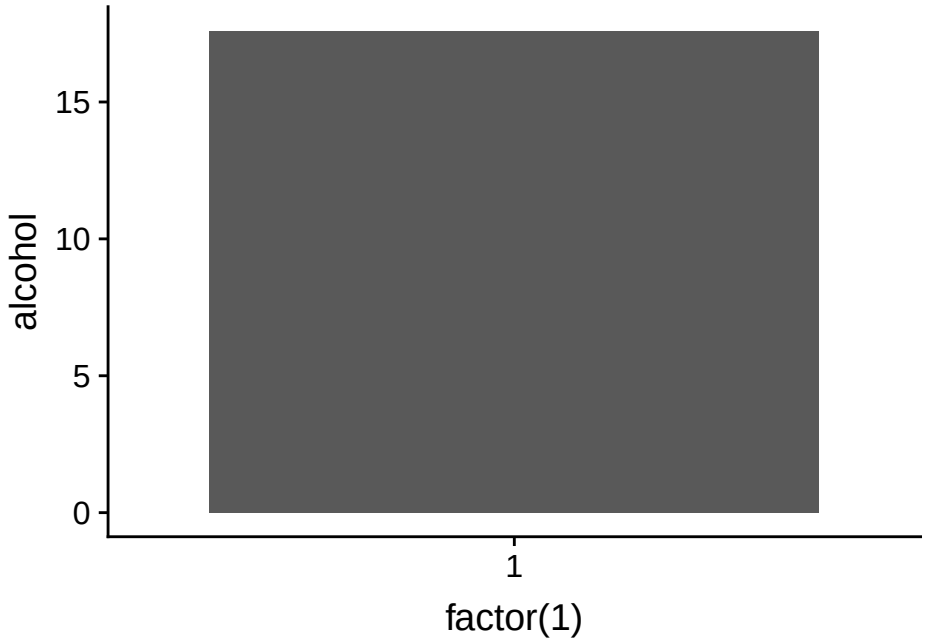
Wobei für `function` verschiedene Funktionen zur Verfügung stehen. Wir wählen hier `y.fun = mean` um den Mittelwert der y-Variable zu bekommen. Für `geom` können alle Geoms von `ggplot2` verwendet werden. Wir wählen hier `"bar"`.

Leider erwartet `geom_bar` sowohl eine abhängige Variable als auch eine Gruppierungsvariable. Wir wenden daher einen kleinen Trick an und definieren einen festen Wert als Gruppierungsvariable: `x = factor(1)`.

```r
lecturer <- read_csv("data/Lecturer_Data.csv") # Lecturer Datensatz importieren
```

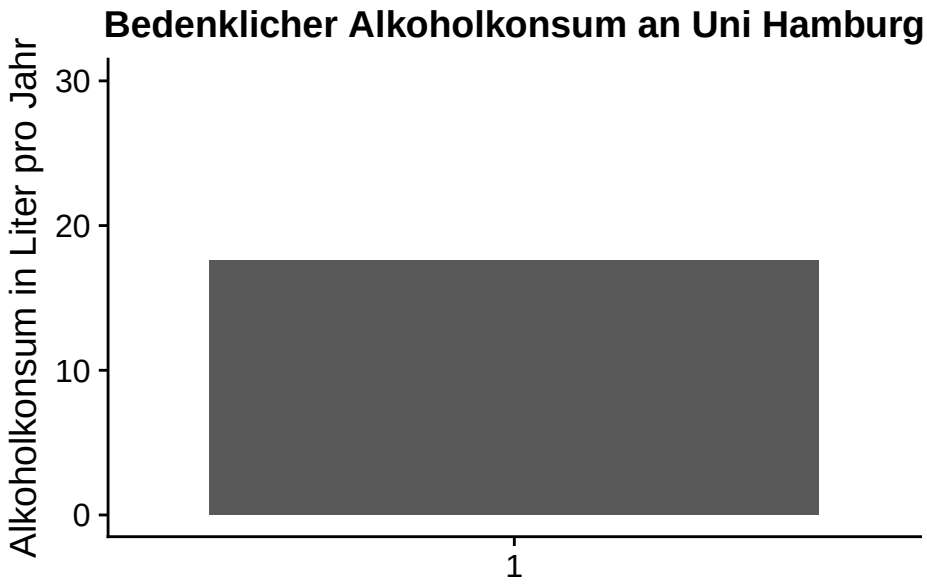
```
## Parsed with column specification:
## cols(
##   ID = col_double(),
##   name = col_character(),
##   birth_date = col_character(),
##   job = col_double(),
##   friends = col_double(),
##   alcohol = col_double(),
##   income = col_double(),
##   neurotic = col_double()
## )
```

```r
bar <- ggplot(lecturer, aes(y = alcohol, x = factor(1))) +
  stat_summary(fun.y = "mean", geom = "bar")
bar
```



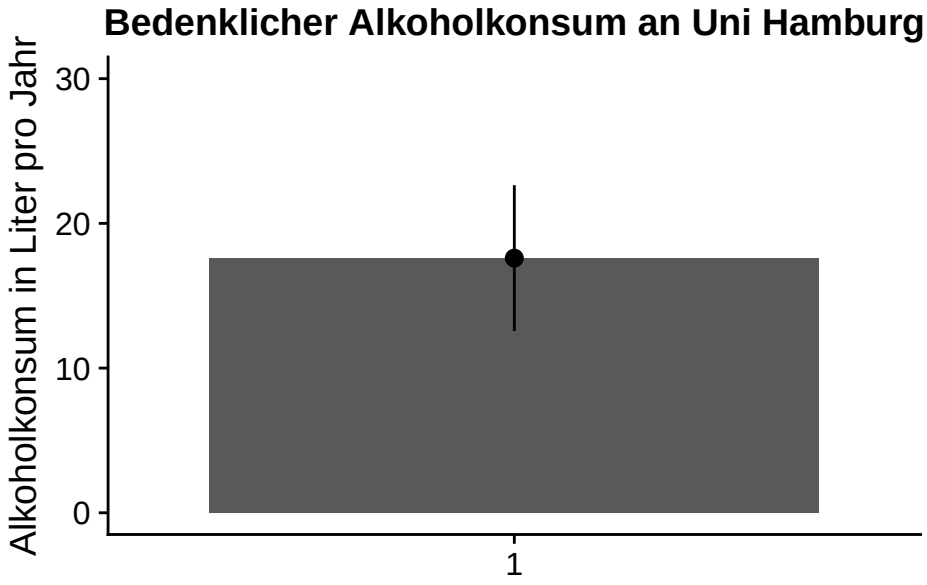
Um die Grafik etwas aussagekräftiger zu gestalten, definieren wir noch die Spannweite von 0 bis zum Maximum der Variable *alcohol*. Hierfür verwenden wir den Layer `scale_y_continuous()`: `scale_y_continuous(limits = c(0, max(lecturer$alcohol)))`. Außerdem ersetzen wir noch die Beschriftung der Achsen und fügen einen Titel hinzu: `labs(x = "", y = "Alkoholkonsum in Liter pro Jahr", title = "Bedenklicher Alkoholkonsum an Uni Hamburg")`

```r
bar <- ggplot(lecturer, aes(y = alcohol, x = factor(1))) +
  stat_summary(fun.y = "mean", geom = "bar") +
  scale_y_continuous(limits = c(0, max(lecturer$alcohol))) +
  labs(x = "",
       y = "Alkoholkonsum in Liter pro Jahr",
       title = "Bedenklicher Alkoholkonsum an Uni Hamburg")
bar
```



Zusätzlich wäre nun noch ein Fehlerbalken interessant, der den 95 %-Konfidenzintervall des Mittelwertes anzeigt. Dazu verwenden wir ebenfalls einen `stat_summary` Layer: `stat_summary(fun.data = mean_cl_normal, geom = "pointrange")`

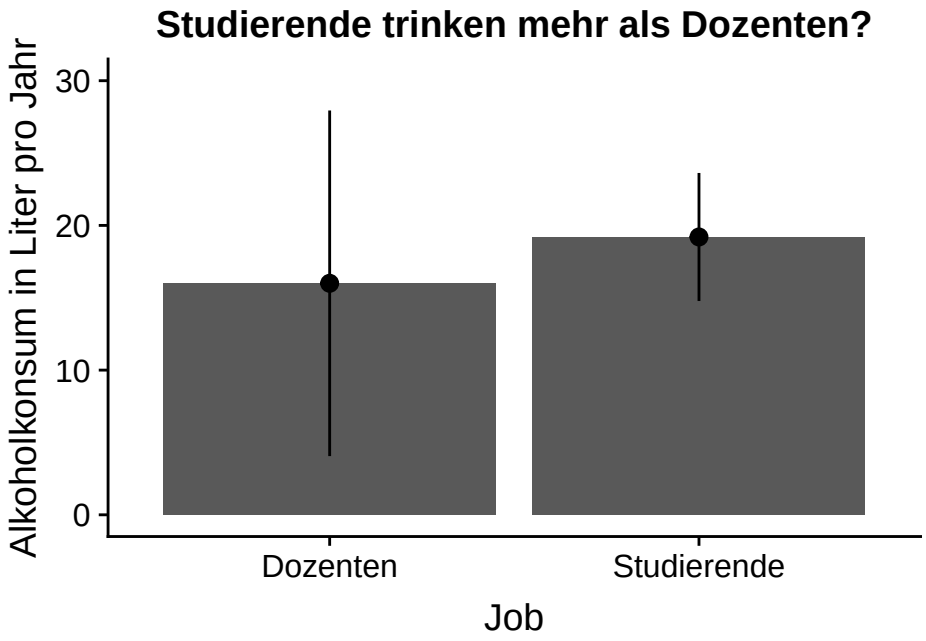
```
bar <- ggplot(lecturer, aes(y = alcohol, x = factor(1))) +
  stat_summary(fun.y = "mean", geom = "bar") +
  stat_summary(fun.data = mean_cl_normal, geom = "pointrange") +
  scale_y_continuous(limits = c(0, max(lecturer$alcohol))) +
  labs(x = "", y = "Alkoholkonsum in Liter pro Jahr",
       title = "Bedenklicher Alkoholkonsum an Uni Hamburg")
bar
```



Da wir ja bereits einen festen Wert für die Gruppierungsvariable festgelegt haben, ist es kein großer Schritt, unterschiedliche Gruppen darzustellen. Wenn wir beispielsweise den Alkoholkonsum von Dozenten und Studierenden unterscheiden wollen, definieren wir die Variable *job* als Gruppierungsvariable. Vorher definieren wir die Gruppierungsvariable allerdings noch als Faktor, damit wir schöne Beschriftungen für die einzelnen Balken bekommen: `lecturer$job <- factor(lecturer$job, labels = c("Dozenten", "Studierende"))`

```
lecturer$job <- factor(lecturer$job, labels = c("Dozenten", "Studierende"))

bar <- ggplot(lecturer, aes(y = alcohol, x = job)) +
  stat_summary(fun.y = "mean", geom = "bar") +
  stat_summary(fun.data = mean_cl_normal, geom = "pointrange") +
  scale_y_continuous(limits = c(0, max(lecturer$alcohol))) +
  labs(x = "Job", y = "Alkoholkonsum in Liter pro Jahr",
       title = "Studierende trinken mehr als Dozenten?")
bar
```



7 Balkendiagramme für mehrere Variablen

Etwas komplizierter wird es, wenn wir mehrere Variablen in einer Grafik darstellen wollen, da wir hierfür die Daten aufbereiten müssen. Gehen wir davon aus, dass wir `alcohol` und `neurotic` gemeinsam darstellen wollen (was natürlich nur Sinn ergibt, wenn diese identisch skaliert sind). Hierfür müssen wir ein neues Objekt generieren, dass in einer Spalte den Variablennamen und in einer anderen Spalte den jeweiligen Wert eines Probanden enthält. Hierfür verwenden wir die Funktion `melt` aus dem Paket `reshape2`. Zusätzlich benötigen wir wieder eine Gruppierungsvariable.

```
install.packages("reshape2", dep = TRUE)
library(reshape2)
```

```
# Variablen auswählen
lecturer3 <- select(lecturer, alcohol, neurotic)

# Gruppierungsvariable erstellen
lecturer3$group = factor(1)

# Datensatz transformieren
```

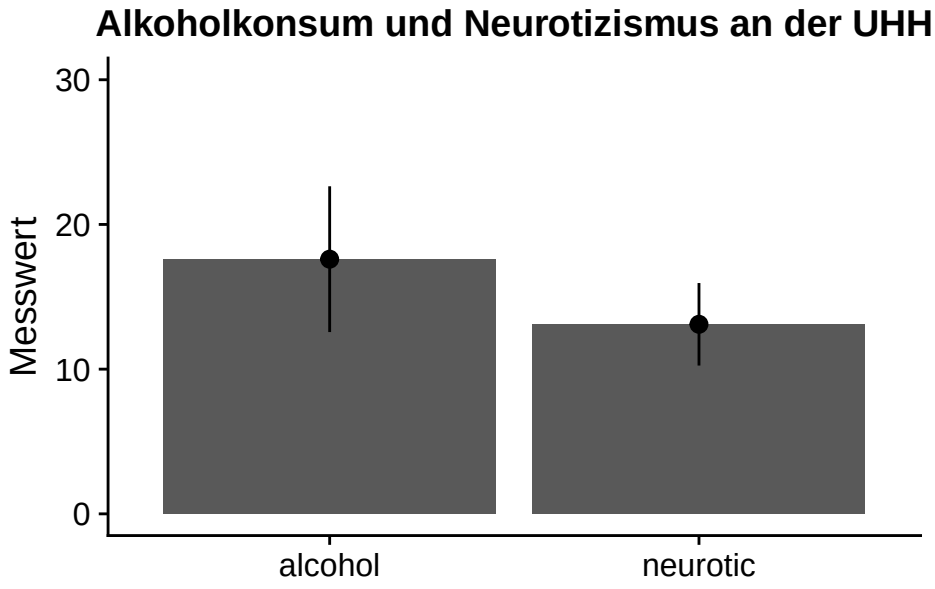
```r
lecturer3_melt <- melt(lecturer3 ,id = "group")

lecturer3_melt
```

```
##    group variable value
## 1      1  alcohol    10
## 2      1  alcohol    15
## 3      1  alcohol    20
## 4      1  alcohol     5
## 5      1  alcohol    30
## 6      1  alcohol    25
## 7      1  alcohol    20
## 8      1  alcohol    16
## 9      1  alcohol    17
## 10     1  alcohol    18
## 11     1 neurotic    10
## 12     1 neurotic    17
## 13     1 neurotic    14
## 14     1 neurotic    13
## 15     1 neurotic    21
## 16     1 neurotic     7
## 17     1 neurotic    13
## 18     1 neurotic     9
## 19     1 neurotic    14
## 20     1 neurotic    13
```

Anschließend können wir die Grafik bauen:

```r
bar <- ggplot(lecturer3_melt, aes(x = variable, y = value)) +
  stat_summary(fun.y=mean, geom = "bar") +
  stat_summary(fun.data = mean_cl_normal, geom = "pointrange") +
  scale_y_continuous(limits = c(0, max(lecturer$alcohol))) +
  labs(x = "", y = "Messwert",
       title = "Alkoholkonsum und Neurotizismus an der UHH")
bar
```



Um auch in dieser Grafik zwischen Dozenten und Studierenden unterscheiden zu können müssen wir dies zunächst bei `melt()` berücksichtigen (`id = "job"`):

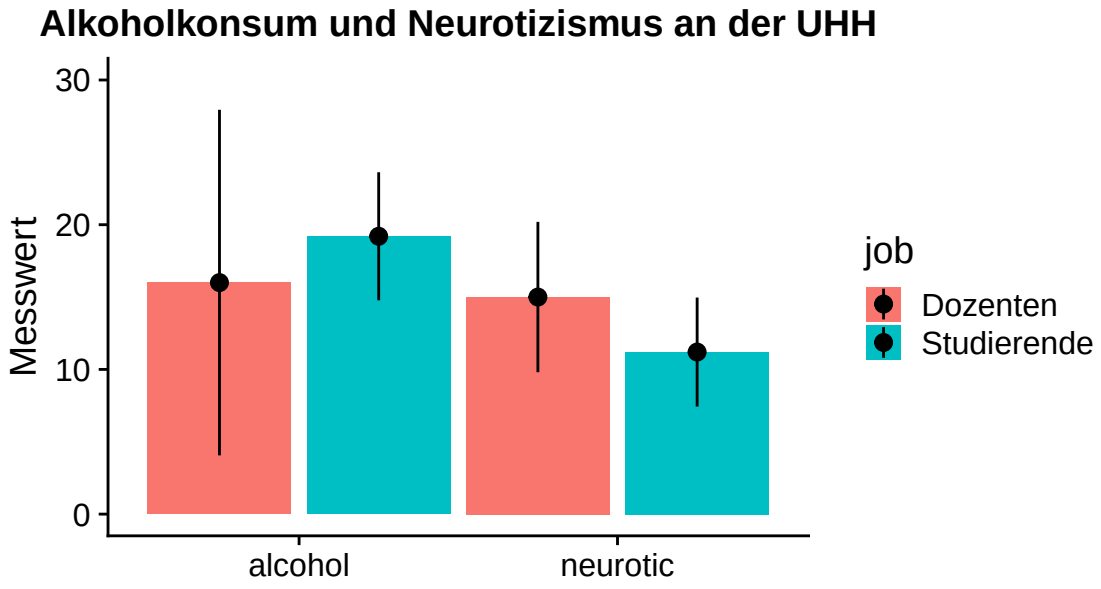
```
# Variablen auswählen
lecturer4 <- select(lecturer, alcohol, neurotic, job)

# Datensatz transformieren
lecturer4_melt <- melt(lecturer4 ,id = "job")
head(lecturer4_melt)
```

```
##           job variable value
## 1    Dozenten  alcohol    10
## 2    Dozenten  alcohol    15
## 3    Dozenten  alcohol    20
## 4    Dozenten  alcohol     5
## 5    Dozenten  alcohol    30
## 6 Studierende  alcohol    25
```

In der Grafik sind nun einige Änderungen notwendig. Zunächst einmal fügen wir zu `ggplot()` ein weiteres Aesthetic hinzu: `fill = job`. Dieses gibt an, dass die Balken entsprechend der Variable „job“ eingefärbt werden. Zusätzlich benötigen wir noch bei den beiden `stat_summary` Layers die Option `position = position_dodge(1)`. Diese sorgt dafür, dass die Balken nicht übereinander, sondern nebeneinander dargestellt werden.

```
bar <- ggplot(lecturer4_melt,
              aes(x = variable, y = value, fill = job)) +
  stat_summary(fun.y = mean, geom = "bar",
               position = position_dodge(1)) +
  stat_summary(fun.data = mean_cl_normal,  geom = "pointrange",
               position = position_dodge(1)) +
  scale_y_continuous(limits = c(0, max(lecturer$alcohol))) +
  labs(x = "",  y = "Messwert",
       title = "Alkoholkonsum und Neurotizismus an der UHH")
bar
```



8 Regressionsergebnisse grafisch darstellen

Regressionsergebnisse müssen nicht immer in einer Tabelle dargestellt werden. Intuitiver ist die Darstellung in einem so genannten Coefficient Plot oder kurz Coefplot. Um nicht händisch die Werte aus dem Regressionsergebnis exportieren und dann plotten zu müssen, nutzen wir das Paket `coefplot`.

```
install.packages("coefplot", dep = TRUE)
library(coefplot)
```

Versuchen wir also das Regressionsergebnis unserer multiplen Regression aus Teil 2 grafisch darzustellen. Hierzu importieren wir zunächst die Daten und schätzen dann die Regression.

```
albums <- read_csv("data/album_sales.csv")
```

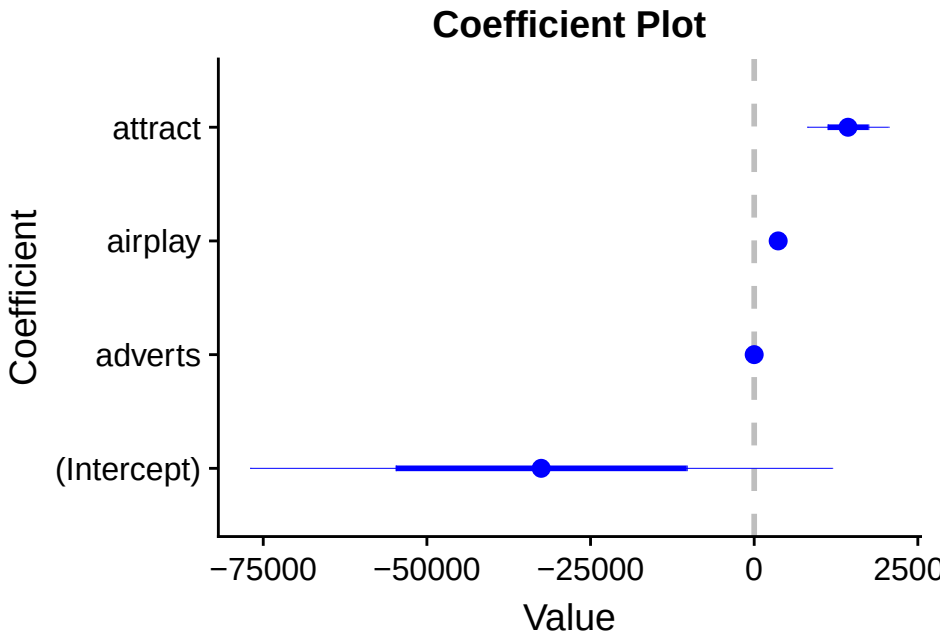
```
## Parsed with column specification:
## cols(
##   adverts = col_double(),
##   sales = col_double(),
##   airplay = col_double(),
##   attract = col_double(),
##   hiphop = col_double()
## )
```

```
model_albums2 <- lm(sales ~ adverts + airplay + attract, data = albums)
summary(model_albums2)
```

```
##
## Call:
## lm(formula = sales ~ adverts + airplay + attract, data = albums)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -156799  -35019   -4632   30155  299239
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.253e+04  2.222e+04  -1.464    0.145
## adverts      5.664e-02  9.365e-03   6.048 7.29e-09 ***
## airplay      3.663e+03  3.551e+02  10.316  < 2e-16 ***
## attract      1.433e+04  3.104e+03   4.618 7.01e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 60070 on 196 degrees of freedom
## Multiple R-squared:   0.51,  Adjusted R-squared:  0.5025
## F-statistic:    68 on 3 and 196 DF,  p-value: < 2.2e-16
```

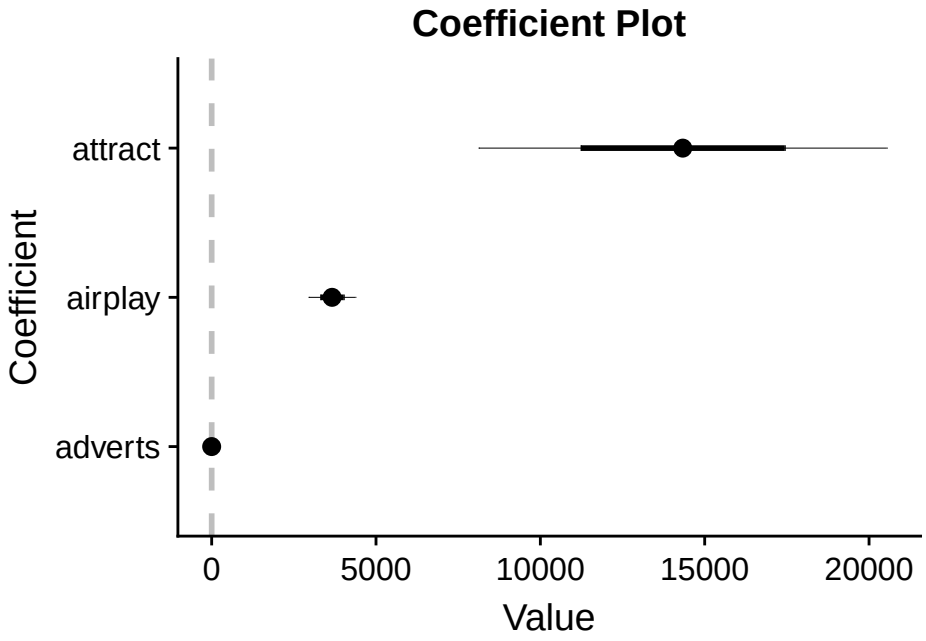
Mit `coefplot` können wir dieses Ergebnis nun grafisch darstellen:

```
coefplot(model_albums2)
```



Um das ganze noch etwas aufzuhübschen, blenden wir die Konstante aus (`intercept = FALSE`) und stellen die Farbe auf schwarz um (`color = "black"`):

```
coefplot(model_albums2,
         intercept = FALSE,
         color = "black")
```



9 Grafiken mit der Maus bearbeiten: ggedit

In RStudio kann man mit dem Paket ggedit auch mit der Maus Hand an eine Grafik legen.

```
install.packages("ggedit", dep = TRUE)
library(ggedit)
```

Hierfür übergibt man eine grafisches Objekt, wie das scatter Objekt, dass wir generiert haben, an die Funktion ggedit().

```
scatter <- ggplot(exam, aes(x = anxiety, y = exam)) +
  geom_point()
ggedit(scatter)
```

Eine ausführliche Anleitung zu ggedit gibt es im „ggedit gitbook“: https://metrumresearchgroup.github.io/ggedit/.